У цьому коді використовуються три основні бібліотеки Python: matplotlib, networkx та matplotlib.colors.

1. **NetworkX**: Це бібліотека для створення, маніпулювання та аналізу структури, динаміки та функцій складних мереж. У цьому коді вона використовується для створення багатократного графа (MultiGraph), додавання країв та ваги до графа.
2. **Matplotlib**: Це бібліотека для створення статичних, анімованих та інтерактивних візуалізацій в Python. У цьому коді вона використовується для візуалізації графа.
3. **Matplotlib.colors**: Це модуль у matplotlib, який використовується для перетворення кольорів між RGB та іншими системами кольорів. У цьому коді він використовується для створення словника кольорів для вузлів графа.
4. Алгоритм для знаходження гамільтонових ланцюгів та циклів адаптовано з сайту [Print all Hamiltonian paths present in a graph | Techie Delight](https://www.techiedelight.com/print-all-hamiltonian-path-present-in-a-graph/)

Особливості роботи цих бібліотек:

* **NetworkX** дозволяє легко створювати графи за допомогою простих команд. Ви можете додавати вузли, краї та ваги до графа. Ви також можете виконувати різні операції з графами, такі як пошук найкоротшого шляху, обчислення центральності вузла тощо.
* **Matplotlib** дозволяє візуалізувати дані на графіках. Ви можете налаштувати різні параметри графіка, такі як кольори, розміри, мітки осей тощо.
* **Matplotlib.colors** дозволяє працювати з кольорами у ваших графіках. Ви можете використовувати його для перетворення кольорів між різними системами кольорів або для створення власних кольорових карт.

Структура графу берется с txt файлу де на кожній лінії 3 цифри вершини які зіднані ребром та вага ребра, в звіті використан граф з роботи 3

Програа проводить розраахунки:

1. Вивести визначення та значення основних характеристик графа (матриця суміжності, інциндентності, хроматичне число та реберне хроматичне число та інші)
2. Написати програму для знаходження всіх ейлерових ланцюгів та циклів
3. Написати програму для знаходження всіх гамільтонових ланцюгів та циклів
4. Реалізувати програмно обхід графа пошуком углиб
5. Реалізувати програмно обхід графа пошуком вшир
6. На мові програмування реалізувани пошук найкородших відстаней на графі від заданої вершини (алгоритм Дейкстри). Вивести схему маршрутів і довжину відстаней.
7. На мові програмування реалізувани пошук найкородших відстаней між будь-якими двома вершинами графу (алгоритм Флойда). Вивести схему маршрутів і довжину відстаней.
8. Знайти між якою парою вершин найкоротша відстань є найдовшою серед всіх найкоротших відстаней.
9. Знайти між якою парою вершин найкоротша відстань є найкоротшою серед всіх найкоротших відстаней.

гамільтоновиі ланцюги

**Програма виводить в консоль:**

Mатриця суміжності:

(0, 1) 6

(0, 2) 8

(0, 3) 3

(0, 4) 3

(1, 0) 6

(1, 2) 8

(1, 3) 4

(2, 0) 8

(2, 1) 8

(2, 4) 6

(3, 0) 3

(3, 1) 4

(3, 4) 4

(3, 5) 3

(4, 0) 3

(4, 2) 6

(4, 3) 4

(5, 3) 3

Mатриця інциндентносi:

(0, 0) 1.0

(1, 0) 1.0

(0, 1) 1.0

(2, 1) 1.0

(0, 2) 1.0

(3, 2) 1.0

(0, 3) 1.0

(4, 3) 1.0

(1, 4) 1.0

(2, 4) 1.0

(1, 5) 1.0

(3, 5) 1.0

(2, 6) 1.0

(4, 6) 1.0

(3, 7) 1.0

(5, 7) 1.0

(3, 8) 1.0

(4, 8) 1.0

Хроматичне число: 3

Реберне хроматичне число: 5

Граф не є ейлеровим

Ребра ейлерового циклу: [(1, 6), (6, 3), (3, 6), (6, 4), (4, 5), (5, 4), (4, 2), (2, 4), (4, 1), (1, 3), (3, 2), (2, 1)]

Граф не має ейлерового шляху.

Оригінальний граф:

[(1, 2), (1, 3), (1, 4), (1, 6), (2, 3), (2, 4), (3, 6), (4, 5), (4, 6)]

Новий ейлерів граф:

[(1, 2), (1, 3), (1, 4), (1, 6), (2, 3), (2, 4), (2, 4), (3, 6), (3, 6), (4, 5), (4, 5), (4, 6)]

Ребра ейлерового шляху: [(1, 2), (2, 3), (3, 1), (1, 4), (4, 2), (2, 4), (4, 5), (5, 4), (4, 6), (6, 3), (3, 6), (6, 1)]

Гамільтоновиі ланцюги:

[1, 2, 3, 6, 4, 5]

[1, 6, 3, 2, 4, 5]

[2, 1, 3, 6, 4, 5]

[2, 3, 1, 6, 4, 5]

[2, 3, 6, 1, 4, 5]

[3, 2, 1, 6, 4, 5]

[3, 6, 1, 2, 4, 5]

[6, 1, 3, 2, 4, 5]

[6, 3, 1, 2, 4, 5]

[6, 3, 2, 1, 4, 5]

[5, 4, 1, 2, 3, 6]

[5, 4, 1, 6, 3, 2]

[5, 4, 2, 1, 3, 6]

[5, 4, 2, 1, 6, 3]

[5, 4, 2, 3, 1, 6]

[5, 4, 2, 3, 6, 1]

[5, 4, 6, 1, 2, 3]

[5, 4, 6, 1, 3, 2]

, 6: [1, 6], 5: [1, 4, 5]}, 2: {2: [2], 1: [2, 1], 3: [2, 3], 4: [2, 4], 5: [2, 4, 5], 6: [2, 4, 6]}, 3: {3: [3], 1: [3, 1], 2: [3, 2], 6: [3, 6], 4: [3, 6, 4], 5: [3, 6, 4, 5]}, 4: {4: [4], 1: [4, 1], 2: [4, 2], 5: [4, 5], 6: [4, 6], 3: [4, 6, 3]}, 6: {6: [6], 1: [6, 1], 3: [6, 3], 4: [6, 4], 2: [6, 4, 2], 5: [6, 4, 5]}, 5: {5: [5], 4: [5, 4], 1: [5, 4, 1], 2: [5, 4, 2], 6: [5, 4, 6], 3: [5, 4, 6, 3]}, 'weight': 'weight'}

Алгоритм Флойда: {1: {1: 0, 2: 6, 3: 8, 4: 3, 6: 3, 5: 6}, 2: {2: 0, 1: 6, 3: 8, 4: 4, 6: 8, 5: 7}, 3: {3: 0, 1: 8, 2: 8, 6: 6, 4: 10, 5: 13}, 4: {4: 0, 1: 3, 2: 4, 5: 3, 6: 4, 3: 10}, 6: {6: 0, 1: 3, 3: 6, 4: 4, 2: 8, 5: 7}, 5: {5: 0, 4: 3, 1: 6, 2: 7, 3: 13, 6: 7}}

Найкоротша відстань є найдовшою серед всіх найкоротших відстаней: (3, 5), 13.

Найкоротша відстань є найкоротшою серед всіх найкоротших відстаней: (1, 4), 3.

